VERSUM: AUDIOVISUAL COMPOSING IN 3D

Tarik Barri

me@tarikbarri.nl

ABSTRACT

This paper introduces the new audiovisual sequencing system "Versum" that allows users to compose in three dimensions. First, the conceptual soil from which this system has sprung is discussed. Secondly, the basic concepts with which Versum operates are explained, providing a general idea of what is meant by sequencing in three dimensions and explaining what compositions made in Versum can look and sound like. Thirdly, the practical ways in which a composer can use Versum to make his own audiovisual compositions are presented by means of a more detailed description of the different graphical user interface elements. Fourthly the consequences of Versum's properties with regard to the composing process are discussed. Then a short description is given of the modular structure of the software underlying Versum. Finally, several foresights regarding the directions in which Versum will continue to develop in the near future are presented.

1. INTRODUCTION

Versum is an audiovisual sequencer. Whereas most sequencers such as Ableton Live and Logic use two-dimensional images to represent sequences of notes and sounds, Versum uses three-dimensional representations and allows the user to sequence in three dimensions. That is, in contrast with the conventional methods of two-dimensional musical notation and execution of the sounds from left to right, the sounds in Versum can be heard in any order that corresponds to a movement that can be made in a three-dimensional space.

For example, the sounds represented in Versum can not only be heard from left to right, they can also be heard from right to left, from up to down, form back to forth, in a spiraling motion, etc. This allows for various compositions in sounds and images, in which the representation of the music becomes as much an aesthetic and important part of the composition as the music itself. When shown to an audience, these representations do not only provide its members with an aesthetical experience, but also enable a deep and intuitive understanding of the compositional structures underlying the music.

2. PHILOSOPHY

2.1. Breaking down barriers

Versum was born out of a need to break down the barriers that usually exist between an electronical musical composer and his audience. I felt that by letting the audience literally see how the composer works and how the composition is constructed, great value could be added to the listening experience. A deeper understanding of the structures underlying a composition would then cause a greater personal sense of engagement. This eventually resulted in the creation of Versum.

Wanting to have the audience understand what's going on in a composition however does not imply that the goal is to make all musical and visual events predictable. On the contrary, having the audience create expectations based upon their understanding of the perceived audiovisual structures can provide the basis for surprises, which are caused by the composer's decision to deny these expectations. Both these expectations themselves and how they are resolved (or not) influence the way people perceive tension in music [1].

2.2. Audiovisual tension

Perceived melodies and rhythmical patterns can instigate expectations in the listener. For example, one might expect the time interval between the last and the next beat to be the same as the one that has been heard between the previous two. Or in a series of tones rising in pitch, it may be expected that the pitch of the next tone will also be higher than the last one. Musical tension can be created when the composer plays with these expectancies, by denying them at some times and realizing them at others. To paraphrase music theorist Leonard Meyer: when expected events do not occur, or when they are delayed, these expectancies produce a state of suspense in the listener. It is this suspense that music cannot do without [2].

What happens in Versum is that the listener has not only his ears to base predictions upon, but also his eyes. As a consequence it is not only auditory, but also visual elements that play a role in the suspense that can be created. Therefore Versum enables the composer to build up a tension that is specifically audiovisual.

In practice, the composer can deny expectations made based upon visual cues in several ways which do not refute the logic of the consistent link between sound and image, but still result in unexpected audiovisual events. The tools with which this can be done are used to build up or resolve audiovisual tensions, in order to construct a specifically audiovisual narrative.

2.3. Spatial relationships

In clinical experimental settings spatial relationships between sounds have frequently been shown to interact with perceived musical attributes in systematic ways [3]. The three-dimensional nature of both Versum's imagery and sounds gives the composer powerful tools to explicitly make use of - and experiment with - the musical implications of spatial distributions of sounds. In a very direct, simple and intuitive manner any complex spatial distribution can be created and altered, with clear visual feedback that enables the composer to easily keep track of the different positions where the sounds are coming from.

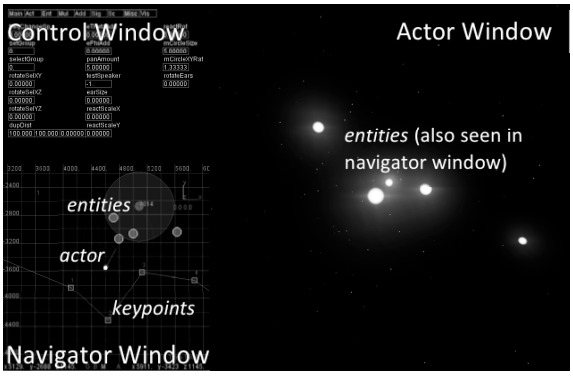

Figure 1. *The three windows generated by Versum, showing spheres.*

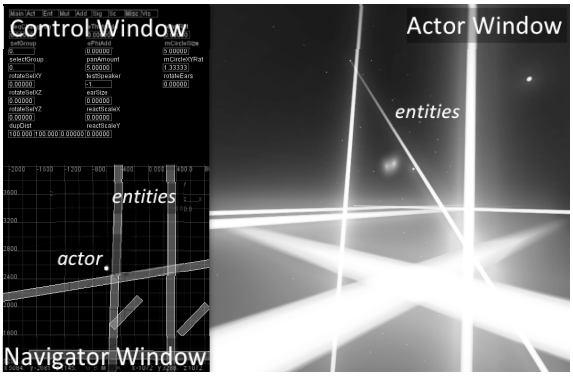

Figure 2. *The three windows generated by Versum, showing lines.*

3. SEEING AND HEARING VERSUM

3.1. *Entities*

After starting up the Versum software, the user sees a depiction of an empty three-dimensional virtual universe that can be populated by audiovisual *entities*. These *entities* are sound generating three-dimensional virtual bodies. There are currently two main types of *entities*, namely *spheres* (see Figure 1) and *lines* (see Figure 2). They come in various colors and sizes and can produce a wide variety of sounds. Exactly how an *entity* sounds and what it looks like is defined by the specific values of its parameters. Many of these parameters influence both the sonic and the visual output, making sound and image analog to each other. For instance, smaller *entities* produce a sound with a smaller amplitude than bigger *entities* do: size in the visual domain is analog to amplitude in the auditory domain. An *entity* depicted with a less smooth surface will produce a more noisy sound (see Figure 3). Likewise, the parameters that determine pitch, filter cutoff frequency and other auditory properties each also have their specific influence in the visual domain.

The user of Versum can place these *entities* anywhere in the virtual universe and can define relationships between them in terms of motion. For instance, *entity A* can have *entity B* orbiting around it, at a speed and radius defined by the user. *Entity B* may also have another two *entities* orbiting around it, and so forth. The ways in which these movements and positions of *entities* influence the perceived sound will be discussed in 3.2 and 3.3.

3.2. The *actor*

What will be heard and seen at any moment in time depends on the position and angle of a moving virtual camera with virtual microphones attached to it. This camera with microphones is called the *actor*. The "Actor Window" on the computer screen shows what the *actor*'s camera sees (see Figure 1 and 2).

When an *actor*'s virtual microphone is in close vicinity of an *entity*, the sound emitted by it will be recorded. If the *actor* would then move further away from the *entity*, the amplitude of the recorded sound would become smaller. All the sounds that are recorded by the *actor'*s microphones are sent to their corresponding output channels.

For instance, let's take an *actor* with two microphones, one to the left and one to the right of the camera. If an *entity* is then situated directly in front of the *actor*'s camera, it will be seen in the center of view and both microphones will record the sound with the same amplitude. The sound of the left microphone is then sent to the left output channel of the sound card and the sound of the right microphone is sent to the right output channel. The sound heard through both speakers will have an equal amplitude. If the *entity* is situated to the left of the *actor*, the left microphone records with a greater amplitude and the sound coming from the left speaker will therefore also have a greater amplitude.

The *actor* may have any amount of microphones that the user requires. If the user wants to use four outputs instead of two, the *actor'*s number of microphones can be set to four, with all of the recorded sounds of these microphones sent to their respective output channels. Theoretically any number of speakers may be used in this manner, placed in front, behind, above and below the listener. The computer's processing speed and the type of sound card are the only limiting factors in this respect.

The *actor* has the ability to move and turn in any direction. If we have several *entities* placed in front of the *actor*'s camera at varying distances, and have the *actor* move forward, it passes these *entities* one by one.

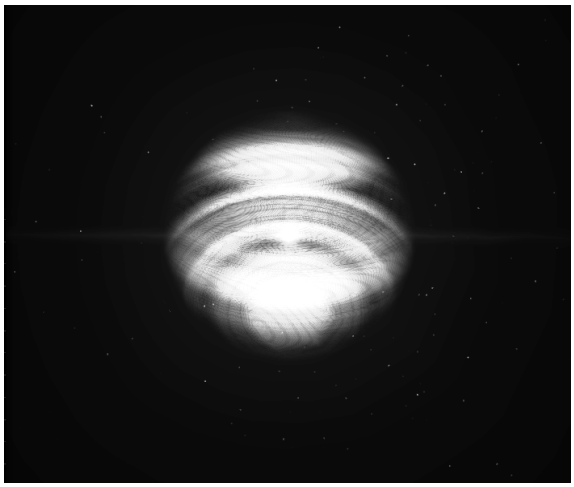

Figure 3. *Example of a noisy entity, as seen in the Actor Window*

Because of the movement of the microphones and changes that this movement causes in the recorded amplitudes of the sounds, a melodic or rhythmical pattern arises. Visually we see the sounds that are still to come at a greater distance from the camera viewpoint, while the sounds of the *entities* that are closer to the camera are already audible. As the *actor* then moves further forward, the *entities* which were previously audible come to lie too far behind for the *actor'*s microphones to hear, and other *entities* which were first seen but not heard, will now be heard.

To give another example, if we have the *actor* facing an *entity* that is being orbited by another, we hear a sound with a static amplitude coming from the *entity* in the center. The rotating *entity* however is heard with a constantly increasing and decreasing amplitude because of the decreasing and increasing distance from the *actor'*s microphones. Having several *entities* rotating the central one at different speeds will in this manner produce complex melodic or rhythmical patterns.

In summary it is the combination of the properties and positions of *entities* and the properties and position of the *actor* that determines the whole auditory and visual experience.

3.3. Sound speed

Versum also has a built in sound speed, meaning that each *entity*'s sound is delayed by a time proportional to the distance it has from each of the *actor*'s microphones. When the *actor* travels past *entities*, the distance of an *entity* to the microphones will decrease or increase with time, and so will the delay time of the sound. When the distance will decrease or increase at high speeds, this changing of delay times will cause a clearly audible Doppler effect, increasing a sense of movement in the listener. At lower speeds the increasing and decreasing of sound delays will cause subtle phasing effects, thus bringing slight variations and liveliness to otherwise more static and monotonous sounds.

4. USING VERSUM

4.1. The Control Window

Versum generates three output windows in total (see Figure 1). The Actor Window, through which the viewpoint of the *actor* can be seen, has already been discussed above. Then there are also two graphical user interface windows: the Control Window and the Navigator Window.

The Control Window contains interface elements which enable the user to set specific parameters of the *entities* such as pitch, noisiness, low pass cutoff frequency, etc. Also the *actor* parameters can be set here, such as the camera angle and the placement of the virtual microphones. General parameters such as the main tempo and the current time can also be adjusted in the Control Window.

4.2. The Navigator Window

The Navigator Window contains a two dimensional map, similar to the kind of map a radar would provide, depicting a portion of the virtual universe, including the positions of the *actor* and the *entities*. The Navigator Window allows the user to zoom in and out, allowing for both very detailed and very broad views of the positions of *entities*, the *actor* and *keypoints* (which will be discussed in 4.3). The view that this map provides can also be rotated for any amount of degrees over any three-dimensional axis to get an ideal view for any editing situation.

Within this map *entities* can be easily added, deleted, copied, dragged and selected by using the mouse and several keyboard shortcuts. Selecting an *entity* within this map by clicking on it with the mouse enables the user to edit its properties in the Control Window. Several functions are also provided for editing the parameters and arranging the positions of large numbers of *entities* at once, so the user can decide for him self whether he wants to compose on a large or small scale.

The Navigator Window also allows the user to quickly and easily move the position and camera angle of the *actor*, thus changing what can be heard and what can be seen in the Actor Window. This function proves to be quite handy for experimenting with the consequences that certain arrangements of *entities* and types of *actor* movements have on the overall composition.

4.3. Keypoints

The freedom to create and change all the *actor* movements, camera angles and *entities* on the spot encourages the user to playfully experiment with the audiovisual structures in space and time that can be created with the software. This freedom on the other hand has the potential disadvantage that it doesn't allow for the desired amount of accuracy in timing, velocity and direction of the *actor'*s movements, since it depends solely on the user's immediate input and is therefore restricted to limitations such as the user's reaction speed. To overcome these limitations, *keypoints* can be used. These provide a way of storing *actor* movements and their timing. They enable the user to edit these movements in minute detail to get a very precise control over an audiovisual composition. The *keypoints* - which can be seen in the Navigator Window - pinpoint exact points in space and time, determining when the *actor* should be where, and which direction the camera should be facing at these points. When an audiovisual composition will then be played from start to end, the *actor* will travel from every *keypoint* to the next in the specified time intervals until the last *keypoint* is reached.

5. CONSEQUENCES FOR COMPOSITIONS

The specific properties of Versum naturally lead to compositions which are quite different from those made in other systems. As a composer I have frequently used Versum to make music and images to investigate what it's effect is on the compositional process.

One of the most important aspects that came up during the experiments with Versum was the fact that this system provides the possibility for creating meta-compositions (see Figure 4): any constellation of *entities* can be heard in any order, coming from any position in space, and heard with any amount of Doppler effect, depending on the movements of the *actor*. This fact has encouraged me often to play certain sequences backwards, diagonally, in a smooth motion, with staccato, jerky movements, etc. The unorthodox sonic and visual results often sparked ideas that I wouldn't have come up with in other ways.

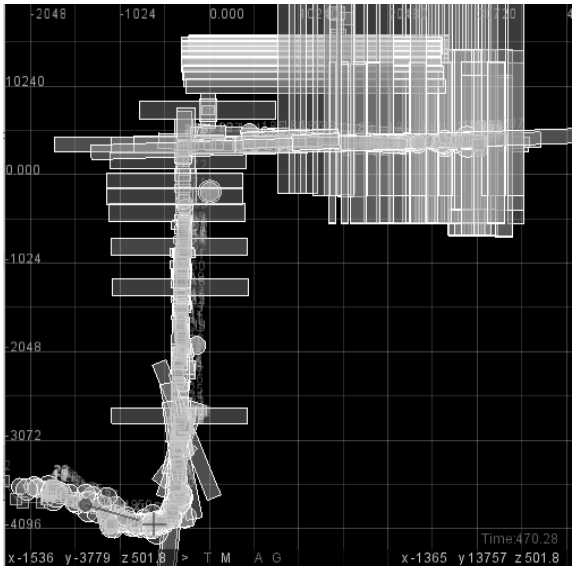

Figure 4. *Example of a composition in Versum, as seen in the Navigator Window.*

Another aspect that plays a big role in the compositional process is that currently all the entities generate continuous, monotonous sounds of which the perceived volume only changes as the *actor* moves closer or further away. In this sense all the sounds are timeless; only our perception of their amplitude changes as the *actor* moves in time. This often results in relatively high fade in and fade out times. In general, as a consequence the compositions have become quite ambient and drone-like, consisting out of continuously altering blends of different sounds coming from continuously changing directions.

Versum's ability to have certain *entities* rotate around other *entities,* which themselves may also rotate around others, has also lead to interesting compositional results. The kind of rhythmical patterns which arise from these rotations are unique to this system, and have also sparked musical and visual ideas that wouldn't have come up otherwise.

6. THE SOFTWARE BEHIND VERSUM

On a software level, Versum consists of four separate pieces of software written in different programming languages, working together as one.

First of all, there is the graphical user interface, visible in the form of the Control- and Navigator Window. This interface is written in Java with heavy use of the Processing library[1].

Secondly, there is the core program, also written in Java. Messages are sent from the interface to this core and back by use of the User Datagram network Protocol (UDP). The core program calculates the time, it keeps track of all the positions of the *entities*, it determines the exact speed at which the *actor* moves, it writes and opens files, etc.

For the three-dimensional graphical output I use a third program made in the Max/MSP/Jitter environment. Via the "mxj" object in Max, messages are sent from the core to the graphical objects within Max, producing the moving images that are presented in the Actor Window.

Finally, for calculating the sounds and sending them to the computer's sound card there is a fourth program written in Supercollider. The core program sends messages to this program via the Open Sound Control network protocol (OSC), telling it what kind of sounds to produce at any moment.

Thanks to the UDP and OSC protocols for the exchange of messages it was easily possible to split the Versum software into these separate specialized pieces of software. Making the software modular in this way has had the great advantage of being able to employ all these programming languages (Java, Supercollider, Max/MSP) specifically for what they do best. None of these languages by themselves would have been sufficient for producing the end result so efficiently.

7. FUTURE DEVELOPMENT

An objective for the near future is to create the possibility of having multiple composers work within the same instance of Versum simultaneously. This would mean that each composer has his own interface and his own *actor* to hear and see the virtual universe with. Thanks to the fact that Versum actually consists of several modules working together, and the fact that these modules communicate via network protocols, there is no necessity for all of them to run on the same computer. One could therefore have multiple interfaces running on separate computers connected to the same instance of Versum via network cables or a wireless network. Composers could then work together on populating a virtual universe with *entities*, creating their own audiovisual constellations or edit those of each other.

When this feature is implemented, Versum's ability to have several *actors* roaming within the same space would mean that audience members could see and hear the same instance of Versum from different auditory and visual perspectives simultaneously, with each *actor* travelling a separate path and generating a unique composition in the process.

8. REFERENCES

[1] Farbood, M. "A Quantitative, Parametric Model of Musical Tension", PhD thesis, MIT, 2006, p. 24.

[2] Meyer, L. B. *Emotion and Meaning in Music*. Chicago: University of Chicago Press, 1956, pp. 26-27.

[3] Bornstein, M. H. *Psychology and its Allied Disciplines, vol 1*. Hillsdale, NJ: Lawrence Erlbaum Associates, 1984, p. 17

[1] See www.processing.org